Lap Time Optimization Under Energetic Constraints

Niccolò Mazzatenta - contact at nmazzatenta@icloud.com

June 2024

Problem Statement

In racing, lap time minimization is the ultimate goal. Nonetheless, performance degradation due to limited energetic resources may arise:

- for a BEV car, maximum extracted energy from the battery may need to be limited to avoid performance derating (overheating, excessive discharge, ...)
- in F1, KERS released energy per lap is limited
- (other energy-related constraints, e.g., dissipated energy by the brakes, tire wear, ...)

In this work, a locally optimal strategy for lap time minimization under energetic constraints is retrieved solving an NLP with CasADi, an open-source tool for nonlinear optimization.

Here it is discussed the case of a car for which the overall tractive energy should be limited to avoid performance degradation.

Optimal Control Problem

While lap time minimization problems can be addressed directly using the total lap time ***T*** as an optimization variable, here a different approach based on path discretization is used.

The problem is discretized along the track path in ***N*** sections, indexed by ***k***.

The objective function is the total lap time calculated as the sum of the ***dt***s along the path.

A dynamic model ***f*** is used to evaluate the velocity at the subsequent section given the current velocity ***v***, the traction force ***F_trac,*** and the braking force ***F_brk***.

System dynamics is included as continuity constraints using a *direct multiple-shooting* method. ***F_trac*** and ***F_brk*** are constrained according to limitation on power unit and braking system. The product between ***F_trac*** and ***F_brk*** is set to be zero to avoid concurrent activation of the inputs.

The speed profile ***v_lim*** and the associated inputs ***F_trac_lim*** and ***F_brk_lim*** represent the maximum achievable performance on the track when energy is not constrained. These are known inputs for the problem.

$$\underset{v_k, F_{\mathrm{trac}}, F_{\mathrm{brk}}}{\text{minimize}} \sum_{k=0}^{N} \frac{\Delta s}{v_k}$$

$$s.t. \quad \begin{aligned} & v_{k+1} = f(v_k, F_{\mathrm{trac}\,k}, F_{\mathrm{brk}\,k}) + w_k \\ & 0 \le v_k \le v_{\mathrm{lim},k} \\ & v_N = v_0 \\ & 0 \le F_{\mathrm{trac}\,k} \le F_{\mathrm{trac\,max}} \\ & F_{\mathrm{trac}\,k}\, v_k \le P_{\mathrm{trac\,max}} \\ & F_{\mathrm{brk\,max}} \le F_{\mathrm{brk}\,k} \le 0 \\ & F_{\mathrm{trac}\,k} F_{\mathrm{brk}\,k} = 0 \\ & E \le E_{\max} \end{aligned} \qquad \text{for } k = 0, 1, \ldots N-1$$

The expended energy ***E*** along the track by the propulsion system is constrained to be lower than a certain threshold ***E_max.***

$$E = \sum_{k=0}^{N} F_{\text{trac } k} \Delta s \leq E_{\max}$$

System Dynamic

System dynamic ***f*** is described by the equations below.

$$v_{k+1} = v_k + a_k\, t_k + w_k = v_k + \left(\frac{F_{\text{trac } k} + F_{\text{brk } k} + F_{\text{drag } k}}{m} \right) \frac{\Delta s}{|v_k|} + w_k$$

$$F_{\text{drag } k} = -\left(A + B\, v_k + C\, v_k^2 + m\, g\, \sin\theta_k \right)$$

where θ_k is the road slope

w_k is the model error

w_k represents the error between the model and the real system. It is calculated so that the model converges to ***v_lim*** provided the inputs ***F_trac_lim*** and ***F_brk_lim*** are used. This ensures that the trivial solution ***{v_lim, F_trac_lim, F_brk_lim}*** is retrieved when the energetic constraint is not active.

$$w_k = v_{\text{lim } k+1} - f\left(v_k, F_{\text{trac lim } k}, F_{\text{brk lim } k}\right)$$

Running the Analysis

The actual optimization problem is set up and solved within the ***call_casadi_laptime_optimization*** function reported at the end of this script. It takes as input a ***data*** structure containing the reference data. Data structure's fields are ***ds, enrg_cons, f0, f1, f2, f_brk_lim, f_brk_max, f_trac_lim, f_trac_max, g, lap_time, m, path_s, pwr_trac_max, road_slope, sectors, v_lim, v_0*** (all fields are in SI units)

The problem is solved for different levels of ***E_max***, and results are analyzed in the following sections.

Below, the solver performance in solving the complete optimization problem (~12500 variables and a similar number of constraints). Note that for subsequent level of ***E_max %*** the solver is initialized at the previous solution to try to improve convergence.

```
%% IPOPT solver output
```

```
Solving for E = 95.0% E_max ...
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  |   4.00ms ( 23.81us)   2.77ms ( 16.51us)       168
       nlp_g  |  26.00ms (154.76us)  23.37ms (139.11us)       168
  nlp_grad_f  |   4.00ms ( 38.46us)   5.29ms ( 50.83us)       104
  nlp_hess_l  |  35.00ms (343.14us)  39.57ms (387.95us)       102
   nlp_jac_g  |  23.00ms (221.15us)  25.95ms (249.49us)       104
       total  |  33.23 s ( 33.23 s)  33.23 s ( 33.23 s)         1
Solving for E = 92.5% E_max ...
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  |   8.00ms ( 27.49us)   4.52ms ( 15.52us)       291
       nlp_g  |  38.00ms (130.58us)  39.41ms (135.42us)       291
  nlp_grad_f  |   7.00ms ( 51.85us)   6.83ms ( 50.57us)       135
  nlp_hess_l  |  57.00ms (428.57us)  53.21ms (400.05us)       133
   nlp_jac_g  |  38.00ms (281.48us)  34.39ms (254.76us)       135
       total  |  41.57 s ( 41.57 s)  41.57 s ( 41.57 s)         1
Solving for E = 90.0% E_max ...
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
```

```
      nlp_f  |   3.00ms ( 25.86us)   2.19ms ( 18.85us)       116
      nlp_g  |  18.00ms (155.17us)  17.68ms (152.44us)       116
 nlp_grad_f  |   2.00ms ( 23.81us)   4.57ms ( 54.46us)        84
 nlp_hess_l  |  30.00ms (365.85us)  33.84ms (412.68us)        82
  nlp_jac_g  |  26.00ms (309.52us)  23.07ms (274.69us)        84
      total  |  24.06 s ( 24.06 s)  24.06 s ( 24.06 s)         1
```

Resulting strategy

The resulting optimal strategy is to *cut* energy consumption at speed peaks by anticipating traction release and delaying braking, effectively letting the car coast. Note that coasting dynamics is affected by road slope, and so are cut durations.

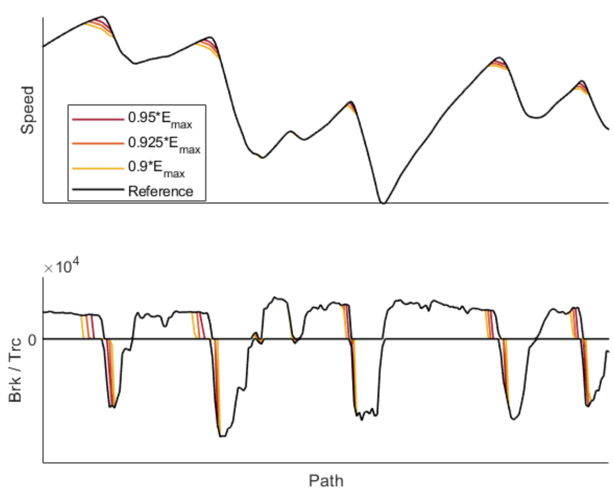

Figure 1. Speed profiles and traction/braking inputs

Cutting at peaks is expected: higher the velocity, lower the time spent in the section while requesting high traction power. This approach lets us quantify optimal cut duration, entity (zero-traction power, indeed!), and estimate the related lap time increment.

Energy	Lap Time Increment
100% E	-
95% E	+0.28s
92.5% E	+0.57s
90% E	+0.96s

Power cuts analysis

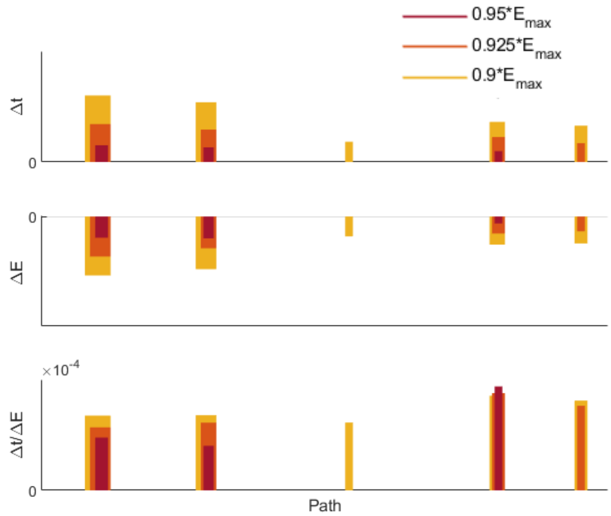

Figure 2. Cumulated time increment (Dt), saved energy (DE), and their ratio (Dt/DE) per each cut. Note the path-dependent relationship between DE and Dt for varying cut durations. The 1st and 2nd cuts lead to higher Dt/DE for increasing durations, while the 4th leads to lower time increment for unit of saved energy. (The 3rd cut for runs at 95% and 92.5% of E_max, while present (see speed and input graphs), is so small that it does not hit a certain significance threshold and so it is not plotted.)

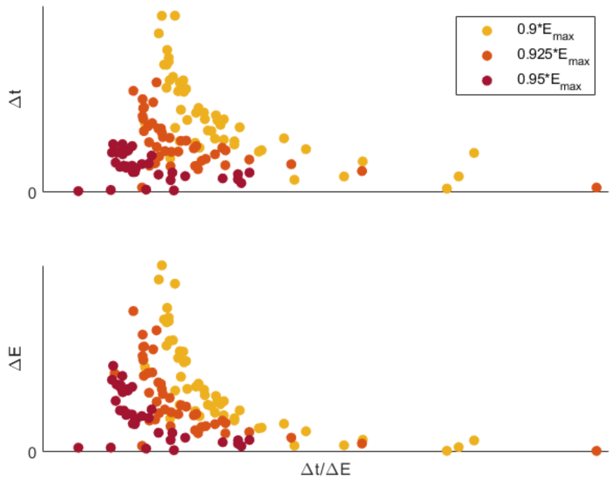

Figure 3. Scatter plot of Dt and DE for all cuts versus Dt/DE. Each point represents a cut. In theory, when increasing the overall saved energy, new cuts should be less efficient than the ones at lower levels of saved energy. According to this, one should expect to see new cuts always on the upper-right side of the others, meaning saving more energy at worse Dt/DE ratio, i.e., increasing even more the lap time. While this is true for most of the cuts, some brighter points seem to appear in regions of previous cuts. This may be explained by the fact that, for the same cut, Dt/DE is not constant but varies with cut duration (see **Figure 2**). In addition, cases as the 3rd cut of the previous graph may contribute as well.

MATLAB function

```
function out = call_casadi_laptime_optimization(data)
%% CASADI SETUP
import casadi.*

steps = numel(data.v_lim);

% Multiple-shooting setup
% Definition of vector variables and vector function
% sysdynmodel is defined below
v_vec = SX.sym('v_k',steps-1,1);
F_trac_vec = SX.sym('F_trac_k',steps-1,1);
F_brk_vec = SX.sym('F_brk_k',steps-1,1);
slope_vec = SX.sym('slope_k',steps-1,1);
comp_vec = SX.sym('comp_k',steps-1,1);

sysdyn = sysdynmodel(v_vec,F_trac_vec,F_brk_vec,slope_vec, comp_vec);

% Single-shooting setup
% Definition of scalar variables and scalar function
v_k = SX.sym('v_k');
F_trac_k = SX.sym('F_trac_k');
F_brk_k = SX.sym('F_brk_k');
slope_k = SX.sym('slope_k');
comp_k = SX.sym('comp_k');

sysdyn_k = sysdynmodel(v_k,F_trac_k,F_brk_k,slope_k,comp_k);

% Model error (i.e., w) is evaluated from multiple-shooting error
vel = sysdyn('v',data.v_lim(1:end-
1),'f_trac',data.f_trac_lim,'f_brk',data.f_brk_lim,'slope',data.road_slope,'comp',D
M.zeros(steps-1,1));
out.v_compensation = data.v_lim(2:end) - full(vel.v_p);

%% MAIN RUN TIME

%% IC from System Dynamic Unroll
% choose a first-trial command vector and unroll the dynamics. This step is not
mandatory, but may improve convergence time as x_0 already
% satisfies all system dynamics constraints

v_0 = sysdynunroll(data.v_0,data.f_trac_0,data.f_brk_0,out.v_compensation);
out.f_trac_k0 = data.f_trac_0;
out.f_brk_k0 = data.f_brk_0;
out.v_0 = v_0;

%% NLP setup
% Variables
```

```
 v = SX.sym('v', steps,1);
 F_trac = SX.sym('f_trac',steps-1,1);
 F_brk = SX.sym('f_brk',steps-1,1);

 X   = [ v             ; F_trac                         ; F_brk
];
 % IC and bounds
 X0  = [ v_0           ; data.f_trac_0                  ; data.f_brk_0
];
 lbw = [ zeros(steps,1); zeros(steps-1,1)               ;
data.f_brk_max*ones(steps-1,1)];
 ubw = [ data.v_lim    ; data.f_trac_max*ones(steps-1,1); zeros(steps-1,1)
];

 % Ideal energy for traction. Trivial to account for power unit efficiency
 E = sum(F_trac.*data.ds);

 % System dynamic evaluation
 sysdyn_kp1 = sysdyn('v', v(1:end-1),'f_trac',F_trac, 'f_brk',
F_brk,'slope',data.road_slope,'comp',out.v_compensation);
 v_p = sysdyn_kp1.v_p;

 % NL Constraints (system dynamic; closed lap; max power; non concurrent inputs;
maximum spendable energy)
 G   = [v_p-v(2:end)          ; v(1)-v(end); F_trac.*v(1:end-1)               ;
F_trac.*F_brk   ; E                 ];
 lbg = [zeros(steps-1,1)      ; 0          ; zeros(steps-1,1)                 ;
zeros(steps-1,1); 0                 ];
 ubg = [zeros(steps-1,1)      ; 0          ; data.pwr_trac_max*ones(steps-1,1);
zeros(steps-1,1); data.enrg_cons_max];

 % Objective
 J = sum(data.ds/v);

 %% NLP definition
 opts.ipopt.print_level=1;
 opts.print_time=1;

 prob = struct('f', J, 'x', X, 'g', G);
 solver = nlpsol('solver', 'ipopt', prob, opts);

 % Solving and retriving solution
 sol = solver('x0', X0, 'lbx', lbw, 'ubx', ubw, 'lbg', lbg, 'ubg', ubg);

 % Save state output
 x_opt = full(sol.x);
 out.v_optim = x_opt(1:steps);
 out.f_trac_optim = x_opt(steps+1:2*steps-1);
 out.f_brk_optim = x_opt(2*steps:end);
```

```matlab
% Save gradients
nlp_grad_f = solver.get_function('nlp_grad_f');
[~,grad_0] = nlp_grad_f(X0,[]);
grad_0 = full(grad_0);
out.grad_v_0 = grad_0(1:steps);
out.grad_f_trac_0 = grad_0(steps+1:2*steps-1);
out.grad_f_brk_0 = grad_0(2*steps:end);

[~,grad_optim] = nlp_grad_f(sol.x,[]);
grad_optim = full(grad_optim);
out.grad_v_optim = grad_optim(1:steps);
out.grad_f_trac_optim = grad_optim(steps+1:2*steps-1);
out.grad_f_brk_optim = grad_optim(2*steps:end);

% Computing additional outputs
out.lap_time = cumtrapz(data.ds,1./out.v_optim); % cumulated lap time
out.cumdlap_time = out.lap_time-data.lap_time; % cumulated delta lap time
out.dlap_time = diff(out.cumdlap_time); % local delta lap time
out.E_local = out.f_trac_optim.*data.ds; % local E traction
out.dE_local = out.E_local-data.f_trac_lim*data.ds; % local delta E traction
out.cumE =cumtrapz(out.E_local); % cumulated E traction
out.cumdE =cumtrapz(out.dE_local); % cumulated delta E traction

%% Sector outputs
% calculate start and stop of power cuts present in optim and not in
% reference
cuts_start = (data.v_lim-out.v_optim)>0.01;
cuts_start_diff = diff(cuts_start);
idxs_cut_start = find(~(cuts_start_diff-1))+1;
cuts_stop = (data.v_lim-out.v_optim)>0.01;
cuts_stop_diff = diff(cuts_stop);
idxs_cut_stop = find(~(cuts_stop_diff+1));

% handle initial/final cuts
if idxs_cut_start(1)>idxs_cut_stop(1)
    idxs_cut_start = [1; idxs_cut_start];
end
if idxs_cut_start(end)>idxs_cut_stop(end)
    idxs_cut_stop = [idxs_cut_stop; numel(data.f_trac_lim)];
end

out.idxs_cut_start = idxs_cut_start;
out.idxs_cut_stop = idxs_cut_stop;
out.path_sector = zeros(numel(idxs_cut_start)*2,1);
out.cumdlap_time_sector = zeros(numel(idxs_cut_start),1);
out.cumdE_sector = zeros(numel(idxs_cut_start),1);

% calculate dt and dE in each cut
```

```
 for i=1:numel(idxs_cut_start)
     if idxs_cut_stop(i)-idxs_cut_start(i)>2
     mask = data.path_s>=data.path_s(idxs_cut_start(i)) &
data.path_s<=data.path_s(idxs_cut_stop(i));
     out.cumdlap_time_sector(i) = trapz(out.dlap_time(mask));
     out.cumdE_sector(i) = trapz(out.dE_local(mask));
     out.path_sector(2*i-1:2*i,1) = [data.path_s(idxs_cut_start(i));
data.path_s(idxs_cut_stop(i))];
     end
 end


 %% NESTED FUNCTIONS
 % System dynamics
     function vdyn_func = sysdynmodel(v_k,F_trac_k,F_brk_k,slope_k,w_k)
         f_drag = data.f0+data.f1.*v_k+data.f2.*v_k.^2 +
data.m*data.g.*sin(slope_k);
         dv = (F_trac_k + F_brk_k - f_drag)./data.m.*data.ds./abs(v_k);
         vkp1 = v_k+dv+w_k;
         vdyn_func =Function('VehDyn',{v_k, F_trac_k,
F_brk_k,slope_k,w_k},{vkp1},{'v','f_trac','f_brk','slope','comp'},{'v_p'});
     end

 % System Dynamics unroll from IC
     function v_unroll = sysdynunroll(v_0,f_trac_k,f_brk_k,w_k)
         v_unroll = DM.zeros(steps,1);
         v_unroll(1) = v_0;
         for step=1:steps-1
             v_kp1
=sysdyn_k('v',v_0,'f_trac',f_trac_k(step),'f_brk',f_brk_k(step),'slope',data.road_s
lope(step),'comp',w_k(step));
             v_unroll(step+1) = v_kp1.v_p;
             v_0 = v_kp1.v_p;
         end
         v_unroll = full(v_unroll);
     end
 end
```